Approximate search with quantized sparse representations

Himalaya Jain[1,2], Patrick Pérez[2], Rémi Gribonval[1], Joaquin Zepeda[2] and Hervé Jégou[1]

[1]Inria Rennes, France
[2]Technicolor, France
[1]{firstname.lastname}@inria.fr, [2]{firstname.lastname}@technicolor.com

Abstract. This paper tackles the task of storing a large collection of vectors, such as visual descriptors, and of searching in it. To this end, we propose to approximate database vectors by constrained sparse coding, where possible atom weights are restricted to belong to a finite subset. This formulation encompasses, as particular cases, previous state-of-the-art methods such as product or residual quantization. As opposed to traditional sparse coding methods, quantized sparse coding includes memory usage as a design constraint, thereby allowing us to index a large collection such as the BIGANN billion-sized benchmark. Our experiments, carried out on standard benchmarks, show that our formulation leads to competitive solutions when considering different trade-offs between learning/coding time, index size and search quality.

Keywords: Indexing, approximate nearest neighbor search, vector quantization, sparse coding.

1 Introduction

Retrieving, from a very large database of high-dimensional vectors, the ones that "resemble" most a query vector is at the heart of most modern information retrieval systems. Online exploration of very large media repositories, for tasks ranging from copy detection to example-based search and recognition, routinely faces this challenging problem. Vectors of interest are abstract representations of the database documents that permit meaningful comparisons in terms of distance and similarity. Their dimension typically ranges from a few hundreds to tens of thousands. In visual search, these vectors are ad-hoc or learned descriptors that represent image fragments or whole images.

Searching efficiently among millions or billions of such high-dimensional vectors requires specific techniques. The classical approach is to re-encode all vectors in a way that allows the design of a compact index and the use of this index to perform fast *approximate search* for each new query. Among the different encoding approaches that have been developed for this purpose, state-of-the-art systems rely on various forms of vector quantization: database vectors are approximated using compact representations that can be stored and searched

efficiently, while the query need not be approximated (asymmetric approximate search). In order to get high quality approximation with practical complexities, the encoding is structured, typically expressed as a sum of codewords stemming from suitable codebooks. There are two main classes of such structured quantization techniques: those based on vector partitioning and independent quantization of sub-vectors [1–3]; those based on sequential residual encoding [4–9].

In this work, we show how these approaches can be taken one step further by drawing inspiration from the sparse coding interpretation of these techniques [10]. The key idea is to represent input vectors as linear combinations of atoms, instead of sums of codewords. The introduction of scalar weights allows us to extend both residual-based and partitioned-based quantizations such that approximation quality is further improved with modest overhead. For this extension to be compatible with large scale approximate search, the newly introduced scalar weights must be themselves encoded in a compact way. We propose to do so by quantizing the vector they form. The resulting scheme will thus trade part of the original encoding budget for encoding coefficients. As we shall demonstrate on various datasets, the proposed quantized sparse representation (i) competes with partitioned quantization for equal memory footprint and lower learning/coding complexity and (ii) outperforms residual quantization with equal or smaller memory footprint and learning/coding complexity.

In the next section, we discuss in more details the problem of approximate vector search with structured quantizers and recall useful concepts from sparse coding. With these tools in hand, we introduce in Section 3 the proposed structured encoding by quantized sparse representations. The different bricks –learning, encoding and approximate search– are presented in Sections 4 and 5, both for the most general form of the framework (residual encoding with non-orthogonal dictionaries) and for its partitioned variant. Experiments are described and discussed in Section 6.

2 Related work

Approximate vector search is a long-standing research topic across a wide range of domains, from communication and data mining to computer graphics and signal processing, analysis and compression. Important tools have been developed around hashing techniques [11], which turn the original search problem into the one of comparing compact codes, *i.e.*, binary codes [12], see [13] for a recent overview on binary hashing techniques. Among other applications, visual search has been addressed by a number of such binary encoding schemes (*e.g.*, [14–18]).

An important aspect of hashing and related methods is that their efficiency comes at the price of comparing only codes and not vectors in the original input space. In the present work we focus on another type of approaches that are currently state-of-art in large scale visual search. Sometimes referred to as *vector compression* techniques, they provide for each database vector $\mathbf{x}$ an approximation $Q(\mathbf{x}) \approx \mathbf{x}$ such that (i) the Euclidean distance (or other related similarity measure such as inner product or cosine) to any query vector $\mathbf{y}$ is well estimated

using $Q(\mathbf{x})$ instead of $\mathbf{x}$ and (ii) these approximate (di)similarity measures can be efficiently computed using the code that defines $Q(\mathbf{x})$.

A simple way to do that is to rely on vector quantization [19], which maps $\mathbf{x}$ to the closest vector in a codebook learned through k-means clustering. In high dimensions though, the complexity of this approach grows to maintain fine grain quantization. A simple and powerful way to circumvent this problem is to partition vectors into smaller dimensional sub-vectors that are then vector quantized. At the heart of product quantization (PQ) [2], this idea has proved very effective for approximate search within large collections of visual descriptors. Different extensions, such as "optimized product quantization" (OPQ) [1] and "Cartesian k-means" (CKM) [3] optimize the chosen partition, possibly after rotation, such that the distortion $\|\mathbf{x} - Q(\mathbf{x})\|$ is further reduced on average. Additionally, part of the encoding budget can be used to encode this distortion and improve the search among product-quantized vectors [20].

It turns out that this type of partitioned quantization is a special case of *structured* or layered quantization:

$$Q(\mathbf{x}) = \sum_{m=1}^{M} Q_m(\mathbf{x}), \tag{1}$$

where each quantizer Q_m uses a specific codebook. In PQ and its variants, these codebooks are orthogonal, making learning, encoding and search efficient. Sacrificing part of this efficiency by relaxing the orthogonality constraint can nonetheless provide better approximations. A number of recent works explore this path.

"Additive quantization" (AQ) [21] is probably the most general of those, hence the most complex to learn and use. It indeed addresses the combinatorial problem of jointly finding the best set of M codewords in (1). While excellent search performance is obtained, its high computational cost makes it less scalable [22]. In particular, it is not adapted to the very large scale experiments we report in present work. In "composite quantization" (CQ) [8], the overhead caused at search time by the non-orthogonality of codebooks is alleviated by learning codebooks that ensure $\|Q(\mathbf{x})\| = \text{cst}$. This approach can be sped up by enforcing in addition the sparsity of codewords [9]. As AQ –though to a lesser extent– CQ and its sparse variant have high learning and encoding complexities.

A less complex way to handle sums of codewords from non-orthogonal codebooks is offered by the greedy approach of "residual vector quantization" (RVQ) [23, 24]. The encoding proceeds sequentially such that the m-th quantizer encodes the *residual* $\mathbf{x} - \sum_{n=1}^{m-1} Q_n(\mathbf{x})$. Accordingly, codebooks are also learned sequentially, each one based on the previous layer's residuals from the training set. This classic vector quantization approach was recently used for approximate search [4, 5, 7]. "Enhanced residual vector quantization" (ERVQ) [4] improves the performance by jointly refining the codebooks in a final training step, while keeping purely sequential the encoding process.

Important to the present work, *sparse coding* is another powerful way to approximate and compress vectors [25]. In this framework, a vector is also approximated as in (1), but with each $Q_m(\mathbf{x})$ being of the form $\alpha_m \mathbf{c}_{k_m}$, where

α_m is a scalar weight and $\mathbf{c}_{k_m}$ is a unit norm *atom* from a learned *dictionary*. The number of selected atoms can be pre-defined or not, and these atoms can stem from one or multiple dictionaries. A wealth of techniques exist to learn dictionaries and encode vectors [25–27], including ones that use the Cartesian product of sub-vector dictionaries [28] similarly to PQ or residual encodings [29, 30] similarly to RQ to reduce encoding complexity. Sparse coding thus offers representations that are related to structured quantization, and somewhat richer. Note however that these representations are not discrete in general, which makes them *a priori* ill-suited to indexing very large vector collections. Scalar quantization of the weights has nonetheless been proposed in the context of audio and image compression [29, 31, 32].

Our proposal is to import some of the ideas of sparse coding into the realm of approximate search. In particular, we propose to use sparse representations over possibly non-orthogonal dictionaries and with vector-quantized coefficients, which offer interesting extensions of both partitioned and residual quantizations.

3 Quantized sparse representations

A sparse coding view of structured quantization Given M codebooks, structured quantization represents each database vector $\mathbf{x}$ as a sum (1) of M codewords, one from each codebook. Using this decomposition, search can be expedited by working at the atom level (see Section 5). Taking a sparse coding viewpoint, we propose a more general approach whereby M dictionaries[1], $C^m = [\mathbf{c}_1^m \cdots \mathbf{c}_K^m]_{D\times K}$, $m = 1 \cdots M$, each with K normalized atoms, are learned and a database vector $\mathbf{x} \in \mathbb{R}^D$ is represented as a linear combination:

$$Q(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m(\mathbf{x}) \mathbf{c}^m_{k_m(\mathbf{x})}, \tag{2}$$

where $\alpha_m(\mathbf{x}) \in \mathbb{R}$ and $k_m(\mathbf{x}) \in [\![1, K]\!]$. Next, we shall drop the explicit dependence in $\mathbf{x}$ for notational convenience. As we shall see in Section 6 (Fig. 1), the additional degrees of freedom provided by the weights in (2) allow more accurate vector approximation. However, with no constraints on the weights, this representation is not discrete, spanning a union of M-dimensional sub-spaces in $\mathbb{R}^D$. To produce compact codes, it must be restricted. Before addressing this point, we show first how it is obtained and how it relates to existing coding schemes.

If dictionaries are given, trying to compute $Q(\mathbf{x})$ as the best ℓ^2-norm approximation of $\mathbf{x}$ is a special case of sparse coding, with the constraint of using exactly one atom from each dictionary. Unless dictionaries are mutually orthogonal, it is a combinatorial problem that can only be solved approximately. Greedy techniques such as projection pursuit [33] and matching pursuit [34] provide particularly simple ways to compute sparse representations. We propose the following

[1] Throughout we use the terminology *codebook* for a collection of vectors, the *codewords*, that can be added, and *dictionary* for a collection of normalized vectors, the *atoms*, which can be linearly combined.

pursuit for our problem: for $m = 1 \cdots M$,

$$k_m = \arg\max_{k \in [\![1,K]\!]} \mathbf{r}_m^\top \mathbf{c}_k^m, \quad \alpha_m = \mathbf{r}_m^\top \mathbf{c}_{k_m}^m, \tag{3}$$

with $\mathbf{r}_1 = \mathbf{x}$ and $\mathbf{r}_{m+1} = \mathbf{r}_m - \alpha_m \mathbf{c}_{k_m}^m$. Encoding proceeds recursively, selecting in the current dictionary the atom with maximum inner-product with the current residual.[2] Once atoms have all been sequentially selected, *i.e.*, the support of the M-sparse representation is fixed, the approximation (2) is refined by jointly recomputing the weights as

$$\hat{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^M} \|\mathbf{x} - C(\mathbf{k})\boldsymbol{\alpha}\|_2^2 = C(\mathbf{k})^\dagger \mathbf{x}, \tag{4}$$

with $\mathbf{k} = [k_m]_{m=1}^M \in [\![1, K]\!]^M$ the vector formed by the selected atom indices, $C(\mathbf{k}) = [\mathbf{c}_{k_1}^1 \cdots \mathbf{c}_{k_M}^M]_{D \times M}$ the corresponding atom matrix and $(\cdot)^\dagger$ the Moore-Penrose pseudo-inverse. Vector $\hat{\boldsymbol{\alpha}}$ contains the M weights, out of KM, associated to the selected support. Note that the proposed method is related to [29, 30].

Learning dictionaries In structured vector quantization, the M codebooks are learned on a limited training set, usually through k-means. In a similar fashion, k-SVD on a training set of vectors is a classic way to learn dictionaries for sparse coding [25]. In both cases, encoding of training vectors and optimal update of atoms/codewords alternate until a criterion is met, starting from a sensible initialization (*e.g.*, based on a random selection of training vectors). Staying closer to the spirit of vector quantization, we also rely on k-means in its spherical variant which fits well our needs: spherical k-means iteratively clusters vector *directions*, thus delivering meaningful unit atoms.

Given a set $\mathcal{Z} = \{\mathbf{z}_1 \cdots \mathbf{z}_R\}$ of R training vectors, the learning of one dictionary of K atoms proceeds iteratively according to:

$$\text{Assignment}: \quad k_r = \arg\max_{k \in [\![1,K]\!]} \mathbf{z}_r^\top \mathbf{c}_k, \ \forall r \in [\![1, R]\!], \tag{5}$$

$$\text{Update}: \quad \mathbf{c}_k \propto \sum_{r : k_r = k} \mathbf{z}_r, \ \|\mathbf{c}_k\| = 1, \ \forall k \in [\![1, K]\!]. \tag{6}$$

This procedure is used to learn the M dictionaries. The first dictionary is learned on the training vector themselves, the following ones on corresponding residual vectors. However, in the particular case where dictionaries are chosen within prescribed mutually orthogonal sub-spaces, they can be learned independently after projection in each-subspace, as discussed in Section 4.

Quantizing coefficients To use the proposed representation for large-scale search, we need to limit the possible values of coefficients while maintaining good approximation quality. Sparse representations with discrete weights have

[2] *Not* maximum absolute inner-product as in matching pursuit. This permits to get a tighter distribution of weights, which will make easier their subsequent quantization.

been proposed in image and audio compression [31, 32], however with scalar coefficients that are quantized independently and not in the prospect of approximate search. We propose a novel approach that serves our aim better, namely employing vector quantization of coefficient vectors $\hat{\boldsymbol{\alpha}}$. These vectors are of modest size, *i.e.*, M is between 4 and 16 in our experiments. Classical k-means clustering is thus well adapted to produce a codebook $A = [\mathbf{a}_1 \cdots \mathbf{a}_P]_{M \times P}$ for their quantization. This is done after the main dictionaries have been learned.[3]

Denoting $p(\boldsymbol{\alpha}) = \operatorname{argmin}_{p \in [\![1,P]\!]} \|\boldsymbol{\alpha} - \mathbf{a}_p\|$ the index of the vector-quantization of $\boldsymbol{\alpha}$ with this codebook, the final approximation of vector $\mathbf{x}$ reads:

$$Q(\mathbf{x}) = C(\mathbf{k})\mathbf{a}_{p(\hat{\boldsymbol{\alpha}})}, \tag{7}$$

with $\mathbf{k}$ function of $\mathbf{x}$ (Eq. 3) and $\hat{\boldsymbol{\alpha}} = C(\mathbf{k})^\dagger \mathbf{x}$ (Eq. 4) function of $\mathbf{k}$ and $\mathbf{x}$.

Code size A key feature of structured quantization is that it provides the approximation accuracy of extremely large codebooks while limiting learning, coding and search complexities: The M codebooks of size K are as expensive to learn and use as a single codebook of size MK but give effectively access to K^M codewords. In the typical setting where $M = 8$ and $K = 256$, the effective number of possible encodings is 2^{64}, that is more than 10^{19}. This 64-bit encoding capability is obtained by learning and using only 8-bit quantizers. Similarly, quantized sparse coding offers up to $K^M \times P$ encoding vectors, which amounts to $M \log_2 K + \log_2 P$ bits. Structured quantization with $M + 1$ codebooks, all of size K except one of size P has the same code-size, but leads to a different discretization of the ambient vector space $\mathbb{R}^D$. The aim of the experiments will be to understand how trading part of the vector encoding budget for encoding jointly the scalar weights can benefit approximate search.

4 Sparse coding extension of PQ and RVQ

In the absence of specific constraints on the M dictionaries, the proposed quantized sparse coding can be seen as a generalization of residual vector quantization (RVQ), with linear combinations rather than only sums of centroids. Hierarchical code structure and search methods (see Section 5 below) are analog. To highlight this relationship, we will denote "Qα-RVQ" the proposed encoder.

In case dictionaries are constrained to stem from predefined orthogonal subspaces V_ms such that $\mathbb{R}^D = \bigoplus_{m=1}^M V_m$, the proposed approach simplifies notably. Encoding vectors and learning dictionaries can be done independently in each subspace, instead of in sequence. In particular, when each subspace is spanned by D/M (assuming M divides D) successive canonical vectors, *e.g.*, $V_1 = \operatorname{span}(\mathbf{e}_1 \cdots \mathbf{e}_{D/M})$, our proposed approach is similar to product quantization (PQ), which it extends through the use of quantized coefficients. We will

[3] Alternate refinement of the vector dictionaries C^ms and of the coefficient codebook A led to no improvement. A possible reason is that dictionaries update does not take into account that the coefficients are vector quantized, and we do not see a principled way to do so.

denote "Qα-PQ" our approach in this specific set-up: all vectors are partitioned into M sub-vectors of dimension D/M and each sub-vector is approximated independently, with one codeword in PQ, with the multiple of one atom in Qα-PQ.

Alg.1: Learning Qα-RVQ

Input: $\mathbf{z}_{1:R}$
Ouput: $C^{1:M}, A$
$\mathbf{r}_{1:R} \leftarrow \mathbf{z}_{1:R}$
for $m = 1 \cdots M$ **do**
  $C^m \leftarrow$ SPHER_K-MEANS$(\mathbf{r}_{1:R})$
  for $r = 1 \cdots R$ **do**
    $k_{m,r} \leftarrow \text{argmax}_{k \in [\![1,K]\!]} \mathbf{r}_r^\top \mathbf{c}_k^m$
    $\mathbf{r}_r \leftarrow \mathbf{r}_r - (\mathbf{r}_r^\top \mathbf{c}_{k_{m,r}}^m)\mathbf{c}_{k_{m,r}}^m$
for $r = 1 \cdots R$ **do**
  $\boldsymbol{\alpha}_r \leftarrow [\mathbf{c}_{k_{1,r}}^1 \cdots \mathbf{c}_{k_{M,r}}^M]^\dagger \mathbf{z}_r$
$A \leftarrow$ K-MEANS$(\boldsymbol{\alpha}_{1:R})$

Alg.2: Coding with Qα-RVQ

Input: $\mathbf{x}, [\mathbf{c}_{1:K}^{1:M}], [\mathbf{a}_{1:P}]$
Output: $\mathbf{k} = [k_{1:M}], p$
$\mathbf{r} \leftarrow \mathbf{x}$
for $m = 1 \cdots M$ **do**
  $k_m \leftarrow \text{argmax}_{k \in [\![1,K]\!]} \mathbf{r}^\top \mathbf{c}_k^m$
  $\mathbf{r} \leftarrow \mathbf{r} - (\mathbf{r}^\top \mathbf{c}_{k_m}^m)\mathbf{c}_{k_m}^m$
$\boldsymbol{\alpha} \leftarrow [\mathbf{c}_{k_1}^1 \cdots \mathbf{c}_{k_M}^M]^\dagger \mathbf{x}$
$p \leftarrow \text{argmin}_{p \in [\![1,P]\!]} \|\boldsymbol{\alpha} - \mathbf{a}_p\|$

Alg.3: Learning Qα-PQ

Input: $\mathbf{z}_{1:R}$
Output: $\tilde{C}^{1:M}, A$
for $r = 1 \cdots R$ **do**
  $[\tilde{\mathbf{z}}_{1,r}^\top \cdots \tilde{\mathbf{z}}_{M,r}^\top] \leftarrow \mathbf{z}_r^\top$
for $m = 1 \cdots M$ **do**
  $\tilde{C}^m \leftarrow$ SPHER_K-MEANS$(\tilde{\mathbf{z}}_{m,1:R})$
  for $r = 1 \cdots R$ **do**
    $k \leftarrow \text{argmax}_{k \in [\![1,K]\!]} \tilde{\mathbf{z}}_{m,r}^\top \tilde{\mathbf{c}}_k^m$
    $\alpha_{m,r} \leftarrow \tilde{\mathbf{z}}_{m,r}^\top \tilde{\mathbf{c}}_k^m$
$A \leftarrow$ K-MEANS$(\boldsymbol{\alpha}_{1:R})$

Alg.4: Coding with Qα-PQ

Input: $\mathbf{x}, [\tilde{\mathbf{c}}_{1:K}^{1:M}], [\mathbf{a}_{1:P}]$
Output: $\mathbf{k} = [k_{1:M}], p$
$[\tilde{\mathbf{x}}_1^\top \cdots \tilde{\mathbf{x}}_M^\top] \leftarrow \mathbf{x}^\top$
for $m = 1 \cdots M$ **do**
  $k_m \leftarrow \text{argmax}_{k \in [\![1,K]\!]} \tilde{\mathbf{x}}_m^\top \tilde{\mathbf{c}}_k^m$
  $\alpha_m \leftarrow \tilde{\mathbf{x}}_m^\top \tilde{\mathbf{c}}_{k_m}^m$
$p \leftarrow \text{argmin}_{p \in [\![1,P]\!]} \|\boldsymbol{\alpha} - \mathbf{a}_p\|$

Learning the dictionaries C^ms and the codebook A for Qα-RVQ is summarized in Alg. 1, and the encoding of a vector with them is in Alg. 2. Learning and encoding in the product case (Qα-PQ) are respectively summarized in Algs. 3 and 4, where all training and test vectors are partitioned in M sub-vectors of dimension D/M, denoted with tilde.

5 Approximate search

Three related types of nearest neighbor (NN) search are used in practice, depending on how the (dis)similarity between vectors is measured in $\mathbb{R}^D$: minimum Euclidean distance, maximum cosine-similarity or maximum inner-product. The three are equivalent when all vectors are ℓ^2-normalized. In visual search, classical descriptors (either at local level or image level) can be normalized in a variety of ways, *e.g.*, ℓ^2, ℓ^1 or blockwise ℓ^2, exactly or approximately.

With cosine-similarity (CS) for instance, the vector closest the query $\mathbf{y}$ in the database $\mathcal{X}$ is $\arg\max_{\mathbf{x}\in\mathcal{X}} \frac{\mathbf{y}^\top \mathbf{x}}{\|\mathbf{x}\|}$, where the norm of the query is ignored for it has no influence on the answer. Considering approximations of database vectors with existing or proposed methods, approximate NN (aNN) search can be conducted without approximating the query (asymmetric aNN [2]):

$$\text{CS} - \text{aNN}: \quad \arg\max_{\mathbf{x}\in\mathcal{X}} \frac{\mathbf{y}^\top Q(\mathbf{x})}{\|Q(\mathbf{x})\|}. \tag{8}$$

As with structured encoding schemes, the form of the approximation in (7) permits to expedite the search. Indeed, for $\mathbf{x}$ encoded by $(\mathbf{k}, p) \in [\![1, K]\!]^M \times [\![1, P]\!]$, the approximate cosine-similarity reads

$$\frac{\mathbf{y}^\top C(\mathbf{k})\mathbf{a}_p}{\|C(\mathbf{k})\mathbf{a}_p\|}, \tag{9}$$

where the M inner products in $\mathbf{y}^\top C(\mathbf{k})$ are among the MK ones in $\mathbf{y}^\top C$, which can be computed once and stored for a given query. For each database vector $\mathbf{x}$, computing the numerator then requires M look-ups, M multiplications and $M-1$ sums. We discuss the denominator below.

In the Qα-PQ setup, as the M unit atoms involved in $C(\mathbf{k})$ are mutually orthogonal, the denominator is equal to $\|\mathbf{a}_p\|$, that is one among P values that are independent of the queries and simply pre-computed once for all. In Qα-RVQ however, as in other quantizers with non-orthogonal codebooks, the computation of

$$\|C(\mathbf{k})\mathbf{a}_p\| = \Big(\sum_{m,n=1}^{M} a_{mp} a_{np} \mathbf{c}_{k_m}^{m\top} \mathbf{c}_{k_n}^{n} \Big)^{1/2} \tag{10}$$

constitutes an overhead. Two methods are suggested in [21] to handle this problem. The first one consists in precomputing and storing for look-up all inter-dictionary inner products of atoms, *i.e.* $C^\top C$. For a given query, the denominator can then be computed with $\mathcal{O}(M^2)$ operations. The second method is to compute the norms for all approximated database vectors and to encode them with a non-uniform scalar quantizer (typically with 256 values) learned on the training set. This adds an extra byte to the database vector encoding but avoids the search time overhead incurred by the first method. This computational saving is worth the memory expense for very large scale systems (See experiments on 1 billion vectors in the next section).

Using the Euclidean distance instead of the cosine similarity, *i.e.*, solving $\arg\min_{\mathbf{x}\in\mathcal{X}} \left\{ \|Q(\mathbf{x})\|^2 - 2\, \mathbf{y}^\top Q(\mathbf{x}) \right\}$ leads to very similar derivations. The performance of the proposed framework is equivalent for these two popular metrics.

6 Experiments

We compare on various datasets the proposed methods, Qα-RVQ and Qα-PQ, to the structured quantization techniques they extend, RVQ and PQ respectively.

We use three main datasets: SIFT1M [35], GIST1M [2] and VLAD500K [36].[4] For PQ and Qα-PQ on GIST and VLAD vectors, PCA rotation and random coordinate permutation are applied, as they have been shown to improve performance in previous works. Each dataset includes a main set to be searched ($\mathcal{X}$ of size N), a training set ($\mathcal{Z}$ of size R) and S query vectors. These sizes and input dimension D are as follows:

dataset	D	R	N	S
SIFT1M	128	100K	1M	10K
GIST1M	960	500K	1M	1K
VLAD500K	128	400K	0.5M	1K

As classically done, we report performance in terms of recall@R, *i.e.*, the proportion of query vectors for which the true nearest neighbor is present among the R nearest neighbors returned by the approximate search.

Introducing coefficients Before moving to the main experiments, we first investigate how the key idea of including scalar coefficients into structured quantization allows more accurate vector encoding. To this end, we compare average reconstruction errors, $\frac{1}{N}\sum_{\mathbf{x}\in\mathcal{X}}\|\mathbf{x}-Q(\mathbf{x})\|_2^2$, obtained on the different datasets by RVQ (resp. PQ) and the proposed approach *before vector quantization of coefficient vector*, which we denote α-RVQ (resp. α-PQ), see Fig. 1. Three structure granularities are considered, $M = 4,\ 8$ and 16. Note that in RVQ and α-RVQ, increasing the number of layers from M to $M' > M$ simply amounts to resuming recursive encoding of residuals. For PQ and α-PQ however, it means considering two different partitions of the input vectors: the underlying codebooks/dictionaries and the resulting encodings have nothing in common.

Reconstruction errors (distortions) are also reported for $K = 2^8$ and 2^{12} respectively. For a given method, reconstruction error decreases when M or K increases. Also, as expected, α-RVQ (resp. α-PQ) is more accurate than RVQ (resp. PQ) for the same (M, K). As we shall see next, most of this accuracy gain is retained after quantizing, even quite coarsely, the coefficient vectors.

Quantizing coefficients Figure 2 shows the effect of this quantization on the performance, in comparison to no quantization (sparse encoding) and to classic structured quantization without coefficients. For these plots, we have used one byte encoding for $\boldsymbol{\alpha}$, *i.e.*, $P = 256$, along with $M \in \{4, 8, 16\}$ and $K = 256$. With this setting, Qα-RVQ (resp. Qα-PQ) is compared to both α-RVQ and RVQ (resp. α-PQ and PQ) with the same values of M and K. This means in particular that Qα-RVQ (resp. Qα-PQ) benefits from one extra byte compared to RVQ (resp. PQ). More thorough comparisons with equal encoding sizes will be the focus of the next experiments. Adding one more byte for RVQ/PQ encoding would significantly increase its learning, encoding and search complexities.

[4] VLAD vectors, as kindly provided by Relja Arandjelović, are PCA-compressed to 128 dimensions and unit ℓ^2-normalized; SIFT vectors are 128-dimensional and have almost constant ℓ^2-norm of 512, yielding almost identical nearest-neighbors for cosine similarity and ℓ^2 distance.

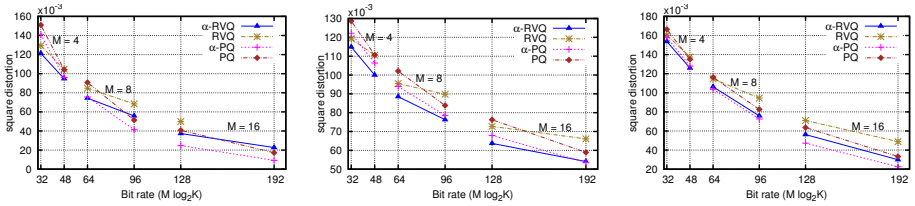


Fig. 1: **Accuracy of structured encoding, with and without coefficients**. Squared reconstruction errors produced by structured encoding (PQ and RVQ) and proposed sparse encoding extensions (α-PQ and α-RVQ). For each method, $M = 4, 8, 16$ and $\log_2 K = 8, 12$ are reported. In absence of coefficient quantization here, each code has $M \log_2 K$ bits, *i.e.* 64 bits for $(M, K) = (8, 256)$.

Since $\boldsymbol{\alpha}$ has M dimensions, its quantization with a single byte gets cruder as M increases, leading to a larger relative loss of performance as compared to no quantization. For $M = 4$, one byte quantization suffices in both structures to almost match the good performance of unquantized sparse representation. For $M = 16$, the increased degradation remains small within Qα-RVQ. However it is important with Qα-PQ: for a small budget allocated to the quantization of $\boldsymbol{\alpha}$, it is even outperformed by the PQ baseline. This observation is counter-intuitive (with additional information, there is a loss). The reason is that the assignment is greedy: while the weights are better approximated w.r.t. a square loss, the vector reconstruction is inferior with Eqn 2. A non-greedy exploration strategy as in AQ would address this problem but would also dramatically increase the assignment cost. This suggests that the size P of the codebook associated with $\boldsymbol{\alpha}$ should be adapted to the number M of layers.

Comparing at fixed code size For large scale search, considering (almost) equal encoding sizes is a good footing for comparisons. This can be achieved in different ways. In the case of RVQ and Qα-RVQ, the recursive nature of encoding provides a natural way to allocate the same encoding budget for the two approaches: we compare Qα-RVQ with (M, K, P) to RVQ with M codebooks of size K and a last one of size P. For PQ and Qα-PQ, things are less simple: adding one codebook to PQ to match the code size of Qα-PQ leads to a completely different partition of vectors, creating new possible sources of behavior discrepancies between the two compared methods. Instead, we compare PQ with M codebooks of size K to Qα-PQ with M dictionaries of size $K/2$ and $P = 2^M$ codewords for coefficient vectors. This way, vector partitions are the same for both, as well as the corresponding code sizes ($M \log_2 K$ bits for PQ and $M \log_2 \frac{K}{2} + \log_2 2^M = M \log_2 K$ bits for Qα-PQ).

Sticking to these rules, we shall compare next structured quantization and quantized sparse representation for equal encoding sizes.

CS-aNN Fig. 3 compares RVQ to Qα-RVQ and PQ to Qα-PQ for different code sizes, from 8 to 24 bytes per vector, on the task of maximum co-

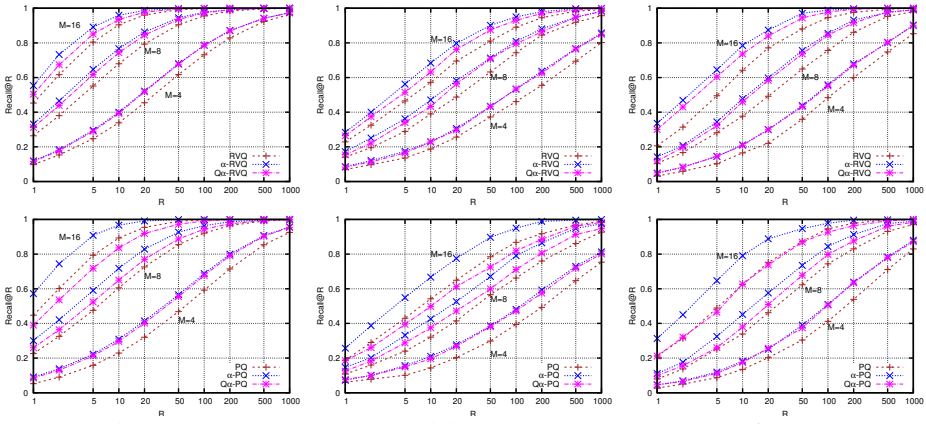


Fig. 2: **Impact of 1-byte α quantization on performance**. Recall@R curves for Qα-RVQ, α-RVQ and RVQ (resp. Qα-PQ, α-PQ and PQ) on the three datasets, with $M \in \{4, 8, 16\}$, $K = 256$ and $P = 256$.

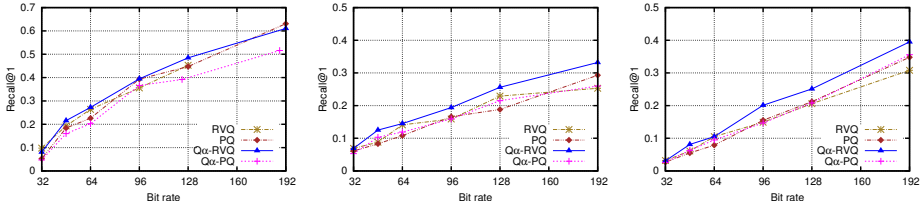
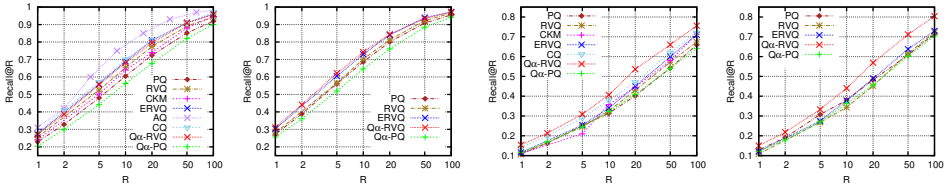


Fig. 3: **Comparative CS-aNN performance for different encoding sizes**. Recall@1 on the three datasets for increasing number of encoding bits, comparing PQ and RVQ with Qα-PQ and Qα-RVQ respectively.

sine similarity over ℓ^2-normalized vectors. Qα-RVQ clearly outperforms RVQ on all datasets, even with a substantial margin on GIST1M and VLAD500K, *i.e.*, around 30% relative gain at 24 bytes. The comparison between PQ and Qα-PQ leads to mixed conclusions: while Qα-PQ is below PQ on SIFT1M, it is slightly above for GIST1M and almost similar for VLAD500K. Note however that, for the same number $M \log_2 K$ of encoding bits, Qα-PQ uses $M\frac{K}{2} + 2^M$ centroids, which is nearly half the number MK of centroids used by PQ in low M regimes (*e.g.*, when $K = 256$, 528 vs. 1024 centroids for $M = 4$ and 1280 vs. 2048 centroids for $M = 8$). Much fewer centroids for equal code size and similar performance yield computational savings in learning and encoding phases.

Euclidean aNN In order to conduct comparison with other state-of-art methods such as extensions of PQ and of RVQ, we also considered the Euclidean aNN search problem, with no prior normalization of vectors. For this problem, the proposed approach applies similarly since the minimization problem



SIFT1M: 64 bits and 72 bits encoding GIST1M: 64 bits and 72 bits encoding

Fig. 4: **Performance comparison for Euclidean-aNN**. Recall@R curves on SIFT1M and GIST1M, comparing proposed methods to PQ, RVQ and to some of their extensions, CKM [3], ERVQ [4], AQ [21] and CQ [8].

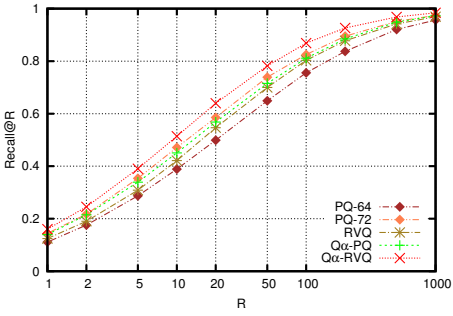
$\text{argmin}_{\mathbf{x}\in\mathcal{X}} \|\mathbf{y}-Q(\mathbf{x})\|^2 = \text{argmax}_{\mathbf{x}\in\mathcal{X}} \mathbf{y}^\top Q(\mathbf{x}) - \frac{\|Q(\mathbf{x})\|^2}{2}$ involves the same quantities as the one in (8).

Recall@R curves are provided in Fig. 4 on two of the three datasets, relying on results reported in [3] for CKM, RVQ and ERVQ, and [21], [8] for AQ and CQ respectively. We observe again that Qα-PQ is below PQ on SIFT but on par with it on GIST. On SIFT, Qα-RVQ, ERVQ and CQ perform similarly, while on GIST Qα-RVQ outperforms all, including CQ and ERVQ. As discussed in Section 2, AQ is the most accurate but has very high encoding complexity. CQ also has higher encoding complexity compared to our simple and greedy approach. For clarity of the figures we have not shown comparison with OPQ [1] which is very close to CKM and performs similarly.[5]

Very large scale experiments on BIGANN We finally conduct very large scale experiments on the BIGANN dataset [6] that contains 1 billion SIFT vectors ($N = 1B$, $R = 1M$ out of the original 100M training set and $S = 10K$ queries). At that scale, an inverted file (IVF) system based on a preliminary coarse quantization of vectors is required. In our experiments, each vector is quantized over $K_c = 8192$ centroids, and it is its residual relative to assigned centroid that is fed to the chosen encoder. At search time, the query is multiply assigned to its $W_c = 64$ closest centroids and W_c searches are conducted over the corresponding vector lists (each of average size N/K_c). Performance is reported in Fig. 5 for PQ, RVQ and their proposed extensions. For all of them the setting is $M = 8$ and $K = 256$, except for PQ-72 bits ($K = 512$). All of them use the exact same IVF structure, which occupies approximately 4GB in memory (4B per vector). For RVQ and Qα-RVQ, norms of approximated database vectors are quantized over 256 scalar values.

The best performance is obtained with the proposed Qα-RVQ approach, which requires 10 bytes per vector, thus a total of 14GB for the whole index. The second best aNN search method is PQ-72 bits, which requires 9 bytes per vector, hence 13GB of index. While both indexes have similar sizes and fit easily

[5] Note also that CKM/OPQ improve on PQ in a way that is complimentary to Qα-PQ. In experiments not reported here, we observed that using OPQ instead of PQ within Qα-PQ gives similar gains as OPQ gives over PQ.



Method (b)	R@1	R@10	R@100	time
PQ-64 (8)	0.111	0.388	0.756	1.00
PQ-72 (9)	0.144	0.471	0.825	1.03
RVQ (9)	0.124	0.421	0.803	1.02
Qα-PQ(9)	0.139	0.450	0.811	1.69
Qα-RVQ(10)	0.160	0.514	0.868	1.72
Qα-RVQ$_{128}$	0.160	0.514	0.868	0.89
Qα-RVQ$_8$	0.151	0.467	0.730	0.17

Fig. 5: **Large scale performance with IVF**. Recall@R on the BIGANN 1B-SIFT dataset and 10K queries. For all methods, $M = 8$ and $K = 256$, except for "PQ-72" ($K = 512$). For quantized sparse coding methods, $P = 256$ and norms in residual variant are quantized over 256 scalar values, resulting encoding sizes (b) being given in bytes per vector. All methods share the same IVF index with $K_c = 2^{13}$ and $W_c = 64$. Subscripted Qα-RVQ denotes variants with additional pruning ($W'_c = 128$ and 8 resp.). Search timings are expressed relative to PQ-64.

in main memory, PQ-72 relies on twice as many vector centroids which makes learning and encoding more expensive.

The superior performance of Qα-RVQ increases the search time by 70% compared to PQ. This can nonetheless be completely compensated for since the hierarchical structure of Qα-RVQ lends itself to additional pruning after the one with IVF. The W'_c atoms most correlated with the query residual in C^1 are determined, and dataset vectors whose first layer encoding uses none of them are ignored. For $W'_c = 128$, the search time is reduced substantially, making Qα-RVQ 10% faster than PQ, with no performance loss (hence superior to PQ-72). A more drastic pruning ($W'_c = 8$) reduces the performance below that of PQ-72, leaving it on par with PQ-64 while being almost 6 times faster.

A variant of IVF, called "inverted multi-index" (IMI) [37] is reported to outperform IVF in speed and accuracy, by using two-fold product quantization instead of vector quantization to produce the first coarse encoding. Using two codebooks of size K_c, one for each half of the vectors, IMI produces K_c^2 inverted lists. We have run experiments with this alternative inverted file system, using $K_c = 2^{14}$ and scanning a list of $T = 100K$, $30K$ or $10K$ vectors, as proposed in [37]. The comparisons with PQ-64 based on the same IMI are summarized in Tab. 1 in terms of recall rates and timings. For all values of T, the proposed Qα-RVQ and Qα-PQ perform the best and with similar search time as RVQ and PQ. Also, Qα-RVQ with $T = 30K$ has the same recall@100 as PQ with $T = 100K$ while being twice as fast (14ms vs. 29ms per query). For a fixed T, PQ and Qα-PQ (resp. RVQ and Qα-RVQ) have the same search speed, as the overhead of finding the T candidates and computing look-up tables dominates for such relatively short lists. The T candidates for distance computation are very finely and scarcely chosen. Therefore, increasing the size K of dictionaries/codebooks

Table 1: **Performance and timings with IMI on 1B SIFTs**. Recalls are reported along with search time in milliseconds per query as a function of the length T of candidate list to be exhaustively scanned. For each method, the encoding size (b) is given in bytes per vector.

	T =100K				T =30K				T =10K			
Method (b)	R@1	R@10	R@100	time	R@1	R@10	R@100	time	R@1	R@10	R@100	time
PQ-64(8)	0.170	0.535	0.869	29	0.170	0.526	0.823	11	0.166	0.495	0.725	5
RVQ(9)	0.181	0.553	0.877	37	0.180	0.542	0.831	14	0.174	0.506	0.729	8
Qα-PQ(9)	0.200	0.587	0.898	30	0.198	0.572	0.848	11	0.193	0.533	0.740	5
Qα-RVQ(10)	0.227	0.630	0.920	37	0.225	0.613	0.862	14	0.217	0.566	0.747	8

in the encoding method directly affects search time. This advocates for our methods, as for equal (M, K) and an extra byte for encoding coefficients, Qα-RVQ and Qα-PQ always give a better performance.

7 Discussion and conclusion

In this work we present a novel quantized sparse representation that is specially designed for large scale approximate NN search. The residual form, Qα-RVQ, clearly outperforms RVQ in all datasets and settings, for equal code size. Within the recursive structure of residual quantization, the introduction of additional coefficients in the representation thus offers accuracy improvements that translate into aNN performance gains, even after drastic vector quantization of these coefficients. Interestingly, the gain is much larger for image level descriptors (GIST and VLAD) which are key to very large visual search. One possible reason for the proposed approach to be especially successful in its residual form lies in the rapid decay of the coefficients that the hierarchical structure induces. In its partitioned variant, this property is not true anymore, and the other proposed approach, Qα-PQ, brings less gain. It does however improve over PQ for image-level descriptors, especially in small M regimes, while using fewer centroids.

As demonstrated on the billion-size BIGANN dataset, the proposed framework can be combined with existing inverted file systems like IVF or IMI to provide highly competitive performance on large scale search problems. In this context, we show in particular that both Qα-PQ and Qα-RVQ offer higher levels of search quality compared to PQ and RVQ for similar speed and that they allow faster search with similar quality. Regarding Qα-RVQ, it is also worth noting that its hierarchical structure allows one to prune out most distant vectors based only on truncated descriptors, as demonstrated on BIGANN within IVF system. Conversely, this nested structure permits to refine encoding if desired, with no need to retrain and recompute the encoding up to the current layer.

On a different note, the successful deployment of the proposed quantized sparse encoding over million to billion-sized vector collections suggests it could help scaling up sparse coding massively in other applications.

References

1. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: CVPR. (2013)
2. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(1) (2011) 117–128
3. Norouzi, M., Fleet, D.: Cartesian k-means. In: CVPR. (2013)
4. Ai, L., Yu, J., Wu, Z., He, Y., Guan, T.: Optimized residual vector quantization for efficient approximate nearest neighbor search. Multimedia Systems (2015) 1–13
5. Chen, Y., Guan, T., Wang, C.: Approximate nearest neighbor search by residual vector quantization. Sensors **10**(12) (2010) 11259–11273
6. Jégou, H., Tavenard, R., Douze, M., Amsaleg, L.: Searching in one billion vectors: re-rank with source coding. In: ICASSP. (2011)
7. Martinez, J., Hoos, H.H., Little, J.J.: Stacked quantizers for compositional vector compression. arXiv preprint arXiv:1411.2173 (2014)
8. Zhang, T., Du, C., Wang, J.: Composite quantization for approximate nearest neighbor search. In: ICML. (2014)
9. Zhang, T., Qi, G.J., Tang, J., Wang, J.: Sparse composite quantization. In: CVPR. (2015)
10. Vedaldi, A., Zisserman, A.: Sparse kernel approximations for efficient classification and detection. In: CVPR. (2012)
11. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC. (1998)
12. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: STOC. (2002)
13. Wang, J., Liu, W., Kumar, S., Chang, S.: Learning to hash for indexing big data - A survey. CoRR (2015)
14. Lv, Q., Charikar, M., Li, K.: Image similarity search with compact data structures. In: CIKM. (2004)
15. Norouzi, M., Punjani, A., Fleet, D.J.: Fast search in hamming space with multi-index hashing. In: CVPR. (2012)
16. Torralba, A., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. In: CVPR. (2008)
17. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for large scale search. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(12) (2012) 2393–2406
18. Xu, H., Wang, J., Li, Z., Zeng, G., Li, S., Yu, N.: Complementary hashing for approximate nearest neighbor search. In: ICCV. (2011)
19. Gersho, A., Gray, R.M.: Vector quantization and signal compression. Volume 159. Springer Science & Business Media (2012)
20. Heo, J.P., Lin, Z., Yoon, S.E.: Distance encoded product quantization. In: CVPR. (2014)
21. Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: CVPR. (2014)
22. Babenko, A., Lempitsky, V.: Tree quantization for large-scale similarity search and classification. In: CVPR. (2015)
23. Barnes, C.F., Rizvi, S., Nasrabadi, N.: Advances in residual vector quantization: a review. IEEE Transactions on Image Processing **5**(2) (1996) 226–262

24. Juang, B.H., Gray, A.J.: Multiple stage vector quantization for speech coding. In: ICASSP. (1982)
25. Elad, M.: Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing. Springer (2010)
26. Mairal, J., Bach, F., Ponce, J., Sapiro, G.: Online learning for matrix factorization and sparse coding. Journal of Machine Learning Research **11** (2010) 19–60
27. Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T.S., Yan, S.: Sparse representation for computer vision and pattern recognition. Proceedings of the IEEE **98**(6) (2010) 1031–1044
28. Ge, T., He, K., Sun, J.: Product sparse coding. In: CVPR. (2014)
29. Zepeda, J., Guillemot, C., Kijak, E.: Image compression using sparse representations and the iteration-tuned and aligned dictionary. IEEE Journal of Selected Topics in Signal Processing (2011)
30. Zepeda, J., Guillemot, C., Kijak, E.: The iteration-tuned dictionary for sparse representations. In: IEEE Workshop on Multimedia Signal Processing. (2010)
31. Frossard, P., Vandergheynst, P., Kunt, M., et al.: A posteriori quantization of progressive matching pursuit streams. IEEE Transactions on Signal Processing **52**(2) (2004) 525–535
32. Yaghoobi, M., Blumensath, T., Davies, M.: Quantized sparse approximation with iterative thresholding for audio coding. In: ICASSP. (2007)
33. Friedman, J.H., Stuetzle, W.: Projection pursuit regression. Journal of the American statistical Association **76**(376) (1981) 817–823
34. Mallat, S.G., Zhang, Z.: Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing **41**(12) (1993) 3397–3415
35. Jégou, H., Douze, M., Schmid, C.: Searching with quantization: approximate nearest neighbor search using short codes and distance estimators. Technical report (2009)
36. Arandjelovic, R., Zisserman, A.: All about VLAD. In: CVPR. (2013)
37. Babenko, A., Lempitsky, V.: The inverted multi-index. In: CVPR. (2012)